MISB Motion Imagery Standards Board	
	MISB ST 1902.1
STANDARD	
Motion Imagery Metadata (MIMD): Model-to-KLV Transmutation Instructions	25 June 2020

1 Scope

The Motion Imagery Metadata (MIMD) documents are composed of the MIMD Modeling Rules, MIMD Model, and Model-to-KLV Transmutation Instructions. MISB ST 1901 [1] defines the modeling rules which are a subset of standardized UML class diagrams specifically minimized to enable transforming, or transmuting, class instances to KLV. MISB ST 1903 [2] defines the MIMD Model, which is a UML model following the rules from MISB ST 1901 and organized as a hierarchy of information including metadata for temporal, platform, payload, sensor, command, automated processes, exploitation, security, and more. This standard defines the Model-to-KLV Transmutation Instructions for constructing bandwidth efficient KLV structures from the model. Other formats such as XML or JSON may have transmutation instructions developed for them in future revisions. This document, in concert with MISB ST 1901, MISB ST 1903 and other MIMD model detail documents, provides a suite of standards embodying the MIMD architecture.

2 References

[1] MISB ST 1901.1 Motion Imagery Metadata (MIMD): Modeling Rules, Jun 2020.

[2] MISB ST 1903.1 Motion Imagery Metadata (MIMD): Model, Jun 2020.

[3] MISB ST 0107.4 KLV Metadata in Motion Imagery, Feb 2019.

[4] SMPTE ST 336:2017 Data Encoding Protocol Using Key-Length-Value.

[5] MISB ST 1904.1 Motion Imagery Metadata (MIMD): Base Attributes, Jun 2020.

[6] ISO/IEC 10646:2014 Information technology – Universal Coded Character Set (UCS).

[7] MISB MISP-2020.1: Motion Imagery Handbook, Oct 2019.

[8] IEEE 754-2008 Standard for Floating-Point Arithmetic [and Floating-Point formats].

[9] MISB ST 1201.4 Floating Point to Integer Mapping, Feb 2019.

[10] MISB ST 1303.2 Multi-Dimensional Array Pack, Jun 2020.

[11] MISB ST 0807.25 MISB KLV Metadata Registry, Jun 2020.

3 Revision History

Revision	Date	Summary of Changes
ST 1902.1	6/25/2020	• Clarified the type of Local Set used by every class transmutation as: BER-OID tags and BER-Length lengths. An equivalent of having byte 6 of a Local Set Universal Label set to 0x0B • Added transmutation for Boolean Type • Added transmutation for Boolean Arrays • Added transmutation for Tuple Type • Changed all MDArray references to use new MDAP notation for ST 1303.2 • Added new key for MIMD Packet, per rules in Section 7, to denote current version is incompatible with previous versions (i.e., not backward compatible) • Added transmutation instructions for Directed Association (reference) and Directed Association arrays

4 Acronyms, Terms, Definitions

APA	Array Processing Algorithm
DLP	Defined Length Pack
IEEE	Institute of Electrical and Electronics Engineers
ISO/IEC	International Organization for Standardization/ International Electrotechnical Commission
KLV	Key Length Value
MDAP	Multi-Dimensional Array Pack
MIMD	Motion Imagery Metadata
MISB	Motion Imagery Standards Board
MISP	Motion Imagery Standards Profile
UML	Unified Modeling Language
VLP	Variable Length Pack
ZLE	Zero Length Element

5 Introduction

The purpose of the MIMD suite of standards is to provide a framework that consolidates and organizes information into a singular, unified metadata model, thereby serving many use cases. This MIMD model then transmutes into different formats, such as KLV or XML.

This MIMD standard addresses the processes for generating efficient KLV (and other future formats) from the MIMD model. Figure 1 identifies the five components of the MIMD system: MIMD Modeling Rules, MIMD Model, MIMD Instance, MIMD Transmutation Instructions, and

MIMD Formats. The blue items represent MISB standards and the green items represent implementation data.

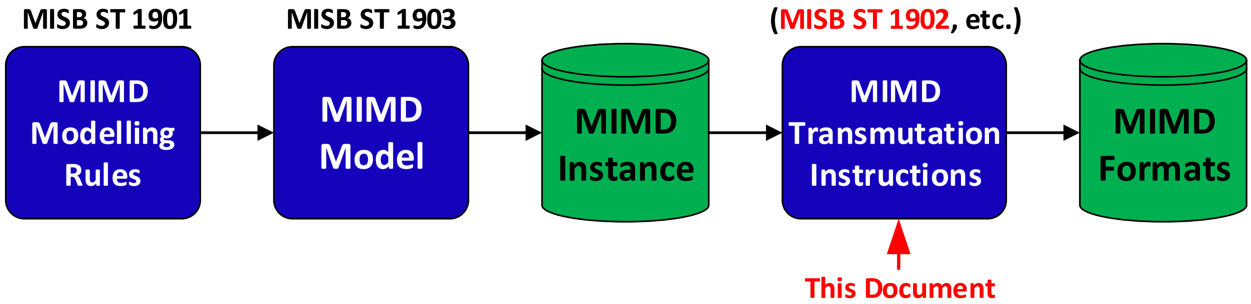


Figure 1: MIMD System Components

The MIMD Modeling Rules comprise a subset of the UML rules for building the MIMD Model. These rules enable transforming, or transmuting, MIMD Instances into MIMD Formats by the MIMD Transmutation Instruction documents.

The MIMD Model is a UML data model following the MIMD Modeling Rules. The Model contains classes, class attributes and defines relationships between classes. MISB ST 1903 and its supporting documents define the MIMD Model.

A MIMD Instance is a data structure implementation of the MIMD Model. While a class defines the allowed contents and rules, an instance is a specific implementation of the class. In other words, a class is the blueprint; the instance is the actual data.

The MIMD KLV Transmutation Instructions provides the "algorithm" for converting a MIMD Model Instance into MIMD Formats. Currently, the only MIMD Format is KLV. MISB ST 1902 defines the MIMD Model-to-KLV Transmutation Instructions that map instances and their relationships to hierarchal KLV structures (e.g., Local Sets and Packs). Additionally, these instructions define methods for the "special" model functions for KLV-unique encodings (e.g., temporal compression). The MISB intends to develop other transmutation instructions for other formats such as XML or JSON as needed.

The MIMD Formats are the final product of the MIMD system. The MIMD KLV Format is a bit-efficient implementation of the MIMD Instance data. Systems generate custom subsets of the MIMD KLV by selecting appropriate classes and class attributes for their system and following the MIMD Model-to-KLV Transmutation Instructions and requirements. Systems that create metadata and transmute it to MIMD Formats are producers. Systems that decode MIMD Formats and process the metadata are receivers.

6 MIMD Model-To-KLV Transmutation Instructions

The MIMD Model-to-KLV Transmutation Instructions are composed of two types: Conventional and MIMD Packet. Conventional instructions (see Section 6.1) provide details on the direct mapping of class instances and their relationships to KLV; most of the MIMD Model will use these instructions to convert to KLV. MIMD Packet instructions (see Section 6.3) provides details on how to create the KLV MIMD Packet and embed the MIMD or Root class instance.

The KLV transmutation processes utilize the instructions and invoke the requirements in MISB ST 0107 [3].

6.1 Conventional Instructions

The Conventional instructions describe how to transmute the MIMD Model class instances to KLV. Each class instance independently transmutes to a KLV Local Set. A Composition relationship defines child classes, which transmute to child KLV Local Sets that embed into their parents Local Set. The child instances may have one or more children, which also transmute to embedded KLV Local sets, etc. This concept defines a recursive transmutation capability for Composition relationships.

Inheritance relationships merge their parent's attributes into the child class and become a single transmuted instance. Figure 2 illustrates a simple MIMD Model and its transmutation to KLV; details of the transmutation methods follow in subsequent sections. In this illustration, the MIMD class, or Root class defines an overall MIMD KLV Packet which is a Defined Length Pack (in blue). The defined length pack contains a check value and the contents of the MIMD class: a version attribute, Platform class relationship, and a Timers class relationship. In a similar fashion, the Platform class is composed of more attributes and class relationships, re-creating the hierarchy of the class model through recursive transmutation.

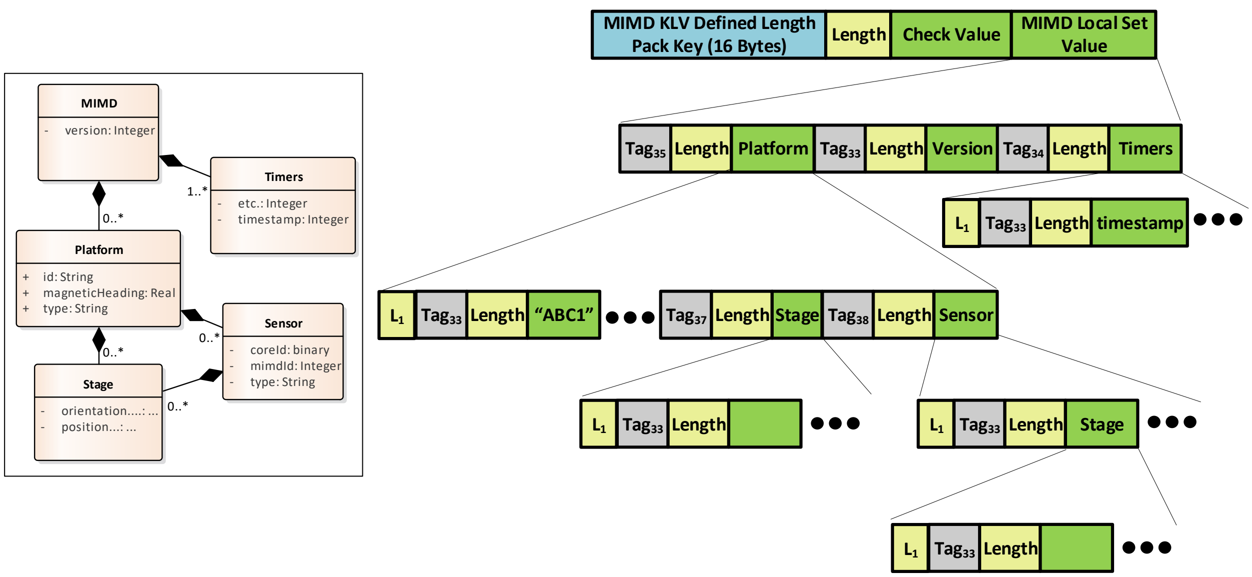


Figure 2:Illustration of Class to KLV Transmutation

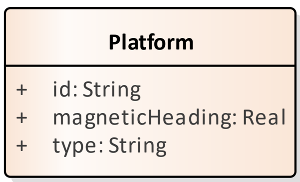
The Conventional instructions include single class instance transmutation (i.e., with no relationships), Composition relationship transmutation, and Inheritance relationship transmutation.

6.1.1 Single Class Instance Transmutation

The MIMD modeling documents (e.g., MISB ST 1903) define classes, attributes within classes, and attribute traits (Name, Type, Identifier, etc.). When mapping class instances to KLV each class becomes a KLV Local Set. All MIMD Local Sets are consistent with a SMPTE 336 [4]

Local Set defining byte 6 of its key equal to 0x0B, which indicates the Length field is ASN.1 BER short or long form and the Local Tag field is encoded using ASN.1 OID BER. Every attribute of a class becomes a Local Set item (see MISB ST 0107 [3]) with tag, length, and value. Section 6.1.1.1 provides details on transmuting attributes to KLV values.

Figure 3 illustrates an example model class and an instance of the class in tabular form with corresponding Local Set item's tags and values. The figure shows a UML Platform class on the left with three attributes (id, magneticHeading, and type) as it would appear in the MIMD Model. The right side shows a class instance in tabular form including associated tags and example values. Local Set Items 1 through 32 are "Reserved for Base Attributes", which are attributes every class includes; MISB ST 1904 [5] provides details for the Base Attributes. In this example, the MISB pre-assigns Local Set Items 33, 34, and 35 to the class attributes within the defining document for the Platform class.



Local Set Items for Platform

Tag	Name	Value
1..32	Reserved for Base Attributes	Not used
33	Id	ABC1
34	magneticHeading	75.3
35	Type	PredatorB

Figure 3: Example of class instantiation

Figure 4 illustrates the result of transforming the class instances of Figure 3 into KLV. The Platform Local Set begins with a tag (*Tag$_{xx}$*) from its parent Local Set, a total Length (*Length (Total)*), and the Local Set items prefixed with Tag 33 through Tag 35. Each Local Set item consists of a Tag (grey), Length (yellow), and Value (green). Each Length shows the value-length in parenthesis.

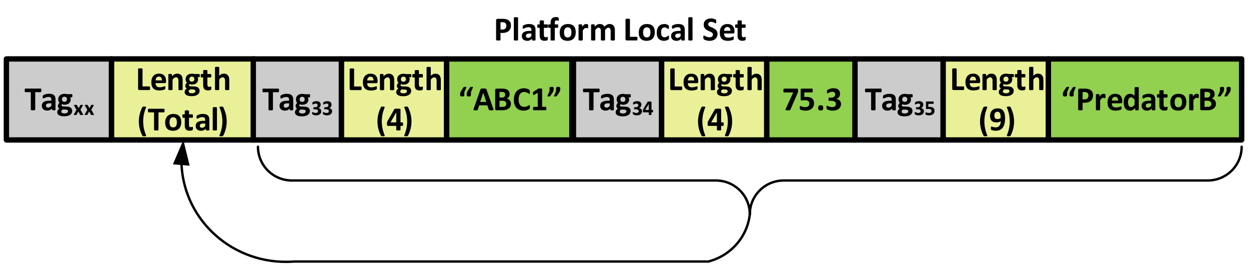


Figure 4: Class Instance to KLV

In cases where a system cannot supply the value for an attribute within its instance, the producer does not include it in the instance, and therefore it is not part of the KLV. For example, if the instance of the Platform class in Figure 3 did not have a value for magneticHeading, the KLV does not include a tag, length, and value for the magneticHeading attribute, as shown in Figure 5.

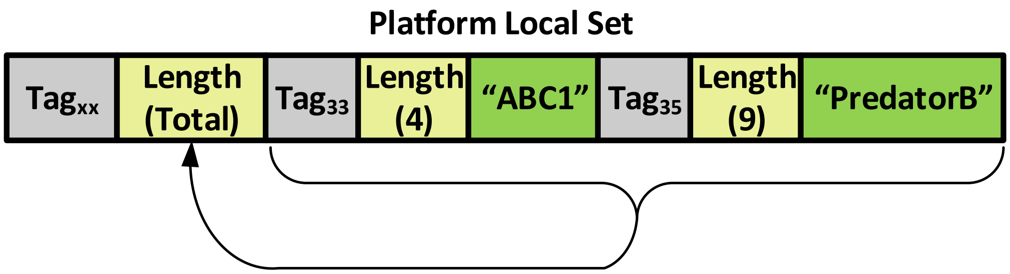


Figure 5: Class Instance to KLV w/o magneticHeading

6.1.1.1 Transmuting Attributes to KLV Local Set Items

An attribute's traits determine its Local Set item's tag and data format. Figure 6 illustrates an attribute's traits (i.e., Name, Type, Identifier, Minimum, Maximum, DefaultResolution, and MaximumLength) and how they transmute to a single KLV Local Set item.

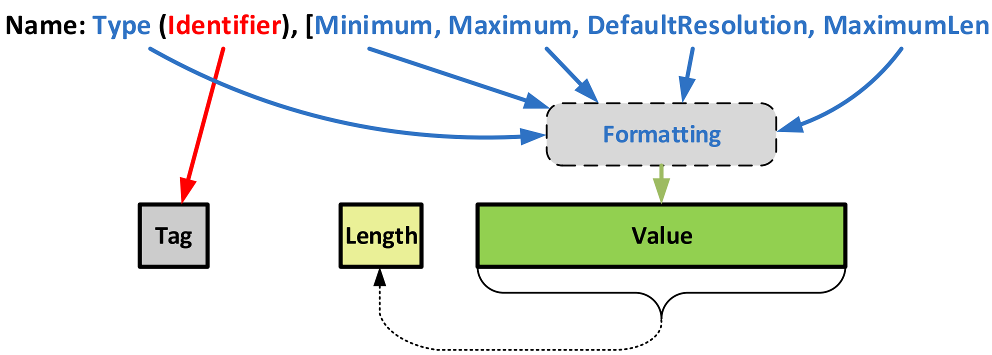


Figure 6: Attribute Traits to KLV Transmutation

A Local Set item's tag is the value of the Identifier trait, encoded as a BER-OID value. The length is the size of the value (in bytes) encoded as a BER short or long form unsigned integer. The item's value is a binary representation of the class attribute's value with the format based on the attribute traits. KLV transmutation does not use the Name, Units, or Deprecate traits.

If an attribute does include a Deprecate trait the producer may not transmute the attribute into KLV.

Requirement	
ST 1902-01	Attributes including the Deprecate trait shall not be transmuted into KLV.

The attribute's traits plus the instance value itself provide the information for transmuting model values to a KLV Local Set value. The process of transmuting values may utilize trait information based on the Type trait. Table 1 lists the traits each data Type may use during transmutation. The Integer and UInt types are qualified by the Minimum and Maximum traits for validating the value is within the given range if provided; these traits do not provide information for Integer or UInt transmutation.

Table 1: Traits Each Type's Transmutation Uses

Type	Minimum	Maximum	DefaultResolution	MaximumLength
String	No	No	No	Yes
Real	Yes	Yes	Yes	No
Integer	Validation Only		No	No
UInt	Validation Only		No	No
Boolean	No	No	No	No

Local Set item values may be in the form of Singular Type or Array Type (see MISB ST 1901). The following sections provide transmutation instructions for a Singular Type. Section 6.1.1.2 provides the transmutation instructions for an Array Type.

6.1.1.1.1 Transmuting String

The MIMD Model values with the attribute type "String" transmutes to a Local Set value by converting the String value into a sequence of UTF-8 characters (see ISO/IEC 10646 [6]). The size, in bytes, of the UTF-8 string becomes the KLV item length. If the attribute includes the MaximumLength trait, the number of characters in the string may not exceed the maximum length defined by the trait. The KLV item length has units of bytes, while the MaximumLength trait has units of characters; developers need to evaluate the correct length. A UTF-8 character has varying length with a minimum of one byte up to a maximum of four bytes; see the Motion Imagery Handbook [7] for further information on "UTF-8" data types.

Requirement	
ST 1902-02	Where a String attribute includes the MaximumLength trait, an encoded UTF-8 character string shall have less than or equal to the traits MaximumLength number of characters.

6.1.1.1.2 Transmuting Real

The MIMD Model attribute type "Real" has two different transmutation formats depending on whether the Minimum and Maximum traits are present. Without these traits, model values of "Real" transmute using the instructions and requirements in MISB ST 0107 for "float" types. Based on the producer's accuracy and bandwidth requirements the producer determines whether to use a single (4 bytes) or double precision (8 bytes) IEEE 754-2008 [8] floating point value[1]. The size, in bytes, of the floating-point value becomes the KLV item length. Receivers use the Local Set item's length to determine if the item's bit pattern is a single or double precision value.

An attribute with the type of "Real" limited by Minimum and Maximum traits, transmutes to MISB ST 1201's [9] IMAP format by following the instructions and requirements in MISB ST 0107 for "IMAPA" or "IMAPB" values. A producer provides the IMAP length or resolution value by either using the DefaultResolution trait, or they may use a different resolution at run-time based on the producer's length or accuracy requirements. The length of the resulting IMAP

[1] Future versions of MISB ST 1902 may include binary16 (half precision) or binary128 (quad precision) from IEEE-754-2008

value becomes the item length of the Local Set item. When receiving the value, the item length, Minimum trait, and Maximum trait values provide the information necessary to decode the IMAP value.

6.1.1.1.3 Transmuting Integer

The MIMD Model attribute type “Integer” transmutes to a Local Set item’s value using the instructions and requirements in MISB ST 0107 for “int” values. The transmutation does not require Minimum or Maximum traits; however, the KLV values must not exceed these limits, if the MIMD Model provides them.

Requirement	
ST 1902-03	Integer attributes that define a valid range with their Minimum and Maximum traits shall have encoded values within the range, inclusively.

The length of the resulting value (i.e., with leading sign extension bytes trimmed) becomes the item length of the Local Set item.

6.1.1.1.4 Transmuting UInt

The MIMD Model attribute type “UInt” transmutes to a Local Set item’s value using the instructions and requirements in MISB ST 0107 for “uint” values.

The transmutation does not require Minimum or Maximum traits; however, the KLV values may not exceed these limits, if the MIMD Model provides them.

Requirement	
ST 1902-04	UInt attributes that define a valid range with their Minimum and Maximum traits shall have encoded values within the range, inclusively.

The length of the resulting value (i.e., with leading zero bytes trimmed) becomes the item length of the Local Set item.

6.1.1.1.5 Transmuting Boolean

The MIMD Model attribute type “Boolean” transmutes to a single byte with either the value of zero (0x00) for false or one (0x01) for true; all other values are not valid. The Local Set’s item length of the resulting value is always one (1).

6.1.1.1.6 Transmuting an Enumeration

An enumeration item transmutes using the value of the enumeration’s numeric identifier. An enumeration numeric identifier (from the enumeration’s table definition) within the MIMD Model transmutes to its Local Set item’s value using the instructions and requirements in MISB ST 0107 for “uint” values (ST 0107 uses a different capitalization for “UInt”). Although the transmutation does not utilize attribute Minimum or Maximum traits, the item’s value needs to match one of the non-deprecated values in the enumeration’s definition table.

Requirement(s)	
ST 1902.1-08	The value of an encoded enumeration numeric identifier shall be a non-deprecated item value within the range of the enumeration’s table definition.

The length of the resulting value (i.e., with leading zero bytes trimmed) becomes the item length of the Local Set item.

6.1.1.1.7 Transmuting a Tuple Type

A Tuple Type is a series of UInt values. The Tuple Type transmutes by serially encoding each element of the series as a BER-OID value. The combination of BER-OID values becomes the KLV item's value. This is an extremely efficient method of encoding a series of UInt values.

An example transmutation of a four element UInt series containing the values (100, 200, 400, 160000) is the combination of the four BER-OID values (0x64, 0x8148, 0x8310, 0x89E200), resulting in a single KLV item's value: 0x6481 4883 1089 E200. The KLV item's length for this example is eight bytes.

The decoding process repeatedly decodes individual BER-OID values from the KLV item's value without exceeding the KLV item's length. The result is a series of individual UInt values which is the Tuple's value.

6.1.1.2 Transmuting an Array Type

As defined in MISB ST 1901, the Array Type is a list of Singular Types arranged as a multi-dimensional array. The MIMD Model may define an Array Type with a specific number of elements (Defined Length Array), maximum number of elements (Maximum Length Array), or a flexible number of elements determined at run-time (Open Length Array). In all cases, systems transmute the Array Type instance into a series of individual Singular Type transmutations that share common encoding parameters, such as length or IMAP parameters. The notation of […] indicates unknown number of dimensions and unknown number of elements in each dimension.

MISB ST 1303 [10] defines the method of transmuting most Array Types to KLV using the Multi-Dimensional Array Pack (MDAP). Data producers determine which Array Processing Algorithm (APA) method, if any, at run-time and every instance may be different for each MIMD Packet and within a Packet.

6.1.1.2.1 Transmuting Real[…]

The MIMD Model attribute type "Real[…]" transmutes into an MDAP (as described in Section 6.1.1.2). Without the Minimum and Maximum traits, each element transmutes into an IEEE 754-2008 element (see Section 6.1.1.1.2). When the model defines the Minimum and Maximum traits, the MDAP APA indicator is set for IMAP and the APA Minimum and Maximum values are set to either the model's Minimum and Maximum trait values or dynamically computed Minimum and Maximum values. The dynamically computed values are not to exceed the models Minimum and Maximum values.

Requirement	
ST 1902-05	Where producers dynamically compute Minimum and Maximum values for an MDAP Array, the computed Minimum and Maximum values shall not exceed the trait's Minimum and Maximum values.

6.1.1.2.2 Transmuting Integer[…]

The MIMD Model attribute type “Integer[...]” transmutes into an MDAP (as described in Section 6.1.1.2). The transmitter pads each element to match the size of the largest element (by byte size) and the appropriate MDAP parameters are set.

6.1.1.2.3 Transmuting UInt[…]

The MIMD Model attribute type “UInt[...]” transmutes into an MDAP (as described in Section 6.1.1.2). Pad each element to match the size of the largest, by byte size, element and the appropriate MDAP parameters are set.

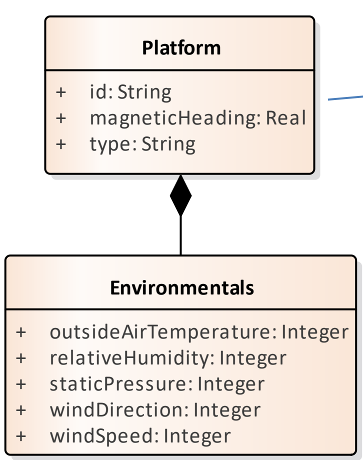
6.1.1.2.4 Transmuting Boolean[…]

The MIMD Model attribute type “Boolean[...]” transmutes into an MDAP (as described in Section 6.1.1.2).

6.1.2 Composition Transmutation

A Composition relationship in a class model becomes a single item in its parent class Local Set; the single item’s value is the “value” portion of a Local Set with its constituent items.

Figure 7 illustrates the transformation of a Composition relationship of two MIMD classes to KLV. The illustration shows two model classes on the left, Platform and Environmentals, and corresponding class instances on the right in tabular format. The Platform class consists of Base Attributes, Platform attributes, and relationship attributes. As in Figure 3 the Base Attributes and Platform attributes map to Local Set items 1 through 35; however, Item 36 maps to the Composition relationship of the Platform class to the Environmentals class. The Platform class is the parent class, so it includes the item for the relationship to the Environmentals child class.



Platform Instance

Tag	Name	Value
1..32	Reserved for Base Attributes	Not used
33	Id	ABC1
34	magneticHeading	75.3
35	Type	PredatorB
36	Environmentals	EMBEDDED

Environmentals Instance

Tag	Name	Value
1..32	Reserved for Base Attributes	Not used
33	outsideAirTemperature	92.0
34	relativeHumidity	50.5
35	staticPressure	15.2
36	windDirection	35.0
37	windspeed	20.0

Figure 7: Illustration of Composition relationship’s class instantiation

When generating KLV from the instances, the Environmentals Local Set embeds into the Platform's Local Set as Item 36. Figure 8 illustrates the resulting KLV generated using the MIMD Model-To-KLV instructions. The Platform Local Set starts with a tag (Tag_{xx}) from its parent Local Set, a Length (Total), and the Local Set items identified by Tag_{33} through Tag_{36}. The Length (Total) is the combined length of all tags, lengths and values for the Platform and its embedded Environmentals Local Set, as shown with the dotted line and arrow. The value for Item 36 is the Environmentals Local Set mapping composed of Environmentals Local Set Items 33 through 37.

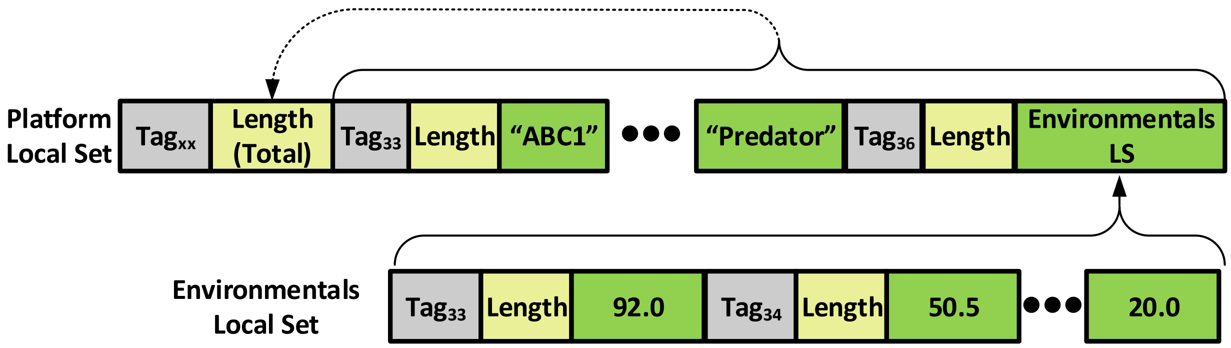


Figure 8:Example of a MIMD Class to KLV Local Set

6.1.2.1 Class Multiplicity

With class multiplicity a parent class relates to zero or more of the same child class. With multiplicity, the parent uses a list of elements where each element in the list is a single instance of the child class.

Figure 9 illustrates the multiplicity relationship of a Platform class with the Sensor class and how it translates to class instances.

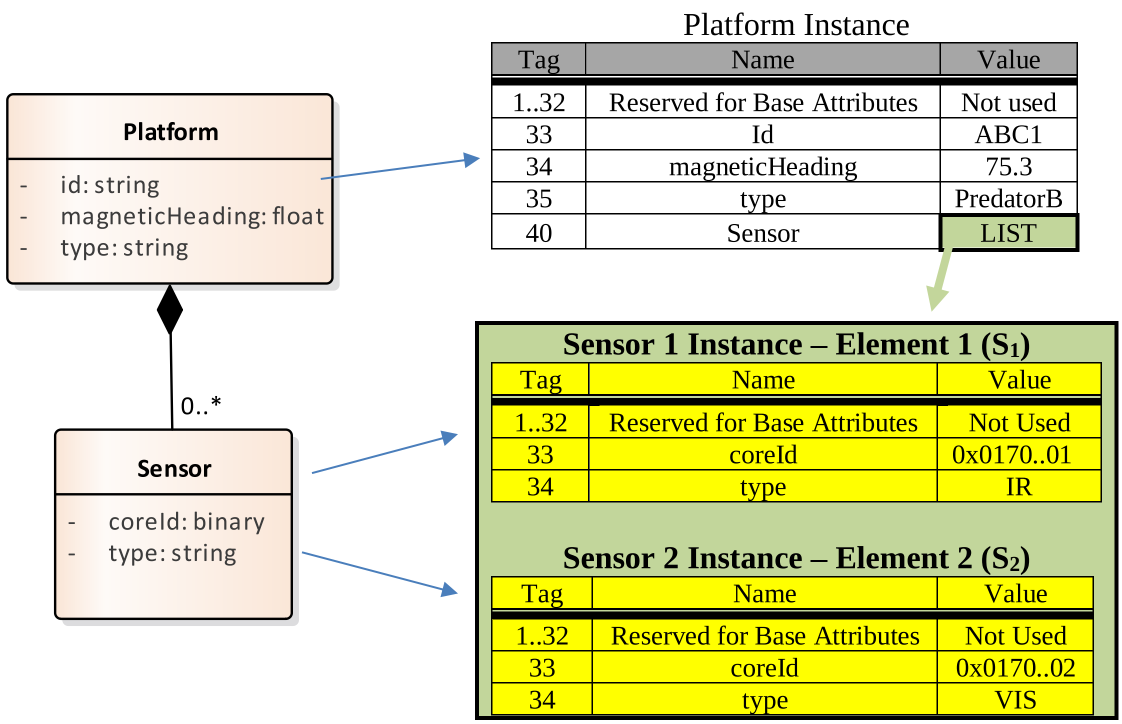


Figure 9: Illustration of Multiplicity

In this example, the Platform class maps to a Platform instance in the same fashion as shown in prior examples; however, the Sensor instantiations are elements of a list within the Platform instantiation, as show in green. The list elements, S_1 and S_2 are not order dependent and they may each have different attributes defined (i.e., their sizes may be different).

Because the list elements vary in size the MIMD Model-To-KLV instructions define the use of a Variable Length Pack (VLP) to encompass the list of child class instances. With a VLP, each item can be of varying length and of any data type; for example, integer, float, or other KLV structures.

The Sensor list converts to a VLP by first transmuting each Sensor instance in the list into a KLV Local Set, then computing its length. Denoting the i^{th} Sensor Local Set as S_i and length of the Sensor Local Set as L_i, a VLP contains a series of length-value pairs; in this case, the length is L_i and the value is S_i.

Figure 10 illustrates the resulting KLV for the Platform class consisting of a multiplicity of Sensors. The Platform Local Set begins with a tag (Tag_{xx}) from its parent Local Set, a total length (*Length (Total)*), and Local Set Items 33, 34, 35 and 40. Item 40 contains the list of Sensor Local Sets in a VLP. The length following Tag_{40} (Item 40's tag), is the size of the complete list including the Sensor Local Sets S_1 and S_2, along with their preceding Lengths, L_1 and L_2 respectively. The illustration shows the value of Item 40 contains a series of list elements. Each element starts with the length of the list element (e.g., L_1) followed by a single Sensor Local Set (e.g., S_1). For example, this illustration shows *Sensor Local Set Element 1* starting with *Length* L_1 then the Sensor Local Set (S_1) elements follow (i.e., Item 33 – 0x0170..01 and Item 34 – "IR"). Preceding each Sensor Local Set (S_i) with its length (L_i) allows decoders to separately extract each list element.

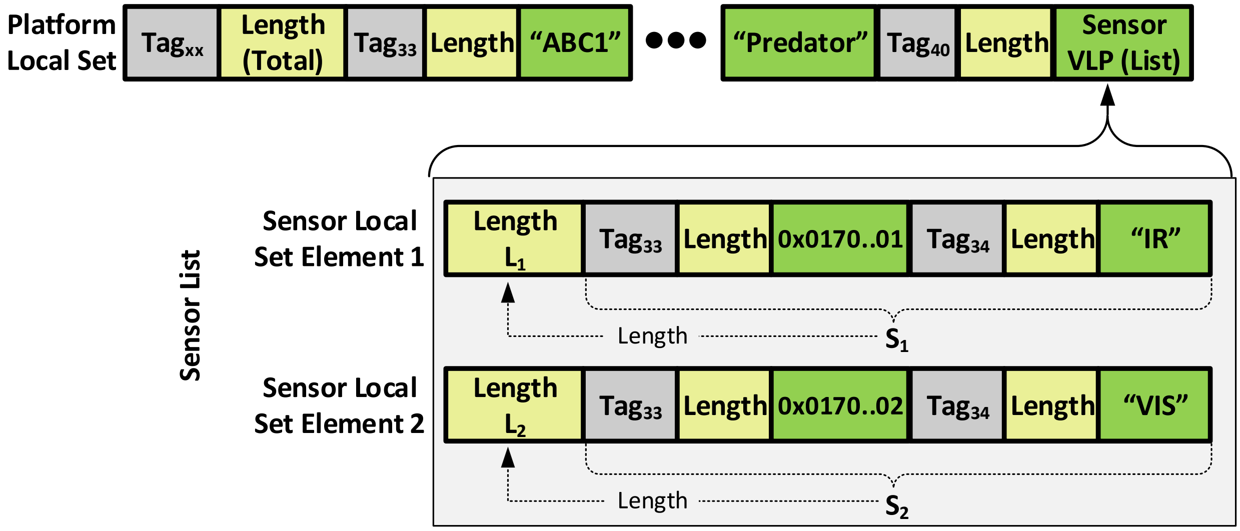


Figure 10: Example of MIMD KLV for a class with multiplicity

6.1.2.1.1 List Organization

MISB ST 1901 defines two methods for organizing Composition relationship lists: Fixed Order and Marked Element. A Composition relationship list of class instances without mimdIds is a Fixed Order list. Transmuting list elements into a VLP maintains the order in the list of class instances. Producers manage the list by adding new elements to the end of the VLP; producers may not insert or delete elements in Fixed Order lists. Producers use a Zero-Length-Element (ZLE) as a filler element to mark an element as unchanged since the last Packet. A ZLE is an element with no Local Set value (e.g., S_i contains no data), so the length (L_i) is zero. Figure 11 illustrates a Sensor VLP List with the first element being a ZLE. In this example Element 1 is a filler value preserving the element position and overall order of the list.

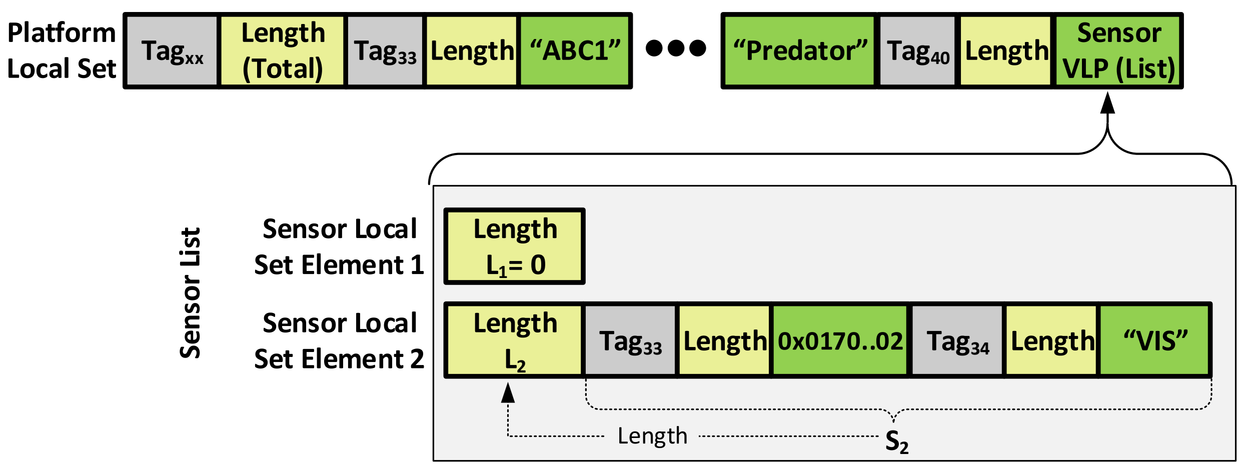


Figure 11: Example of MIMD KLV VLP List with a Zero-Length-Element (ZLE)

A Composition relationship list where all list elements include a mimdId is a Marked Element list. The elements in a Marked Element List may be in any ordered desired and not all elements need to be present in every packet. This offers flexibility but consumes more bytes.

6.1.3 Inheritance Transmutation

An Inheritance relationship in a class model becomes a single item in the parent class's Local Set; the single item's value is the "value" portion of a Local Set with its constituent items. This is the same method as a Composition relationship (see Section 6.1.2), except an Inheritance relationship does not allow multiplicity and the parent only allows one inherited child per parent instance. An Inheritance relationship allows a child class to add more specificity to the more generalized parent. A model which includes two or more child classes inheriting from a parent provides an "exclusive or" relationship between the children; either the parent class is using child class 1, or child class 2, etc.

Figure 12 illustrates inheritance transmutation.

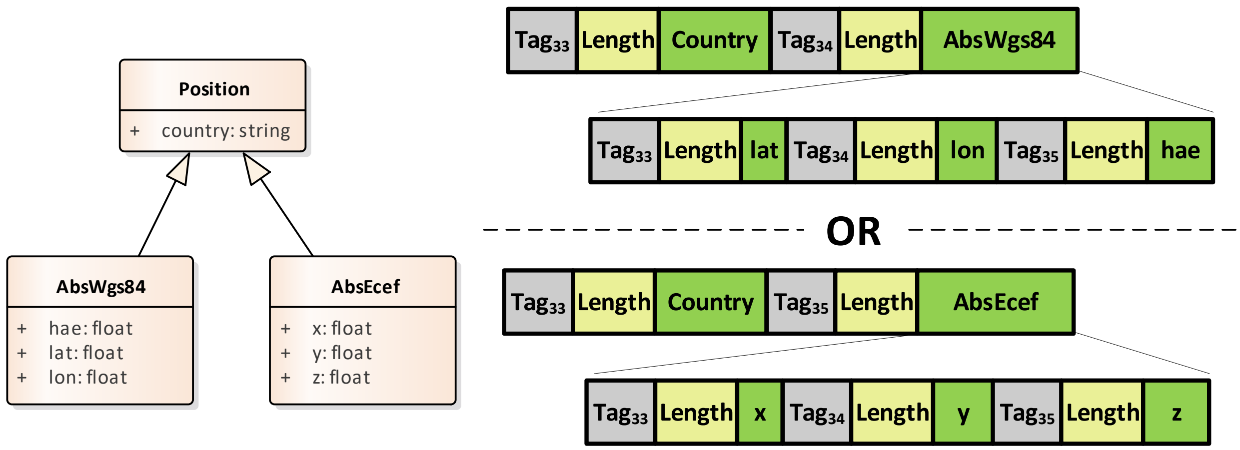


Figure 12: Illustration of Inheritance

The model on the left shows the Position class is the superclass for both child classes AbsWgs84 and AbsEcef. For a given instance of Position producers may include only one child in the Position Local Set. The right side of the figure shows the two possible Local Sets for Position with an inherited child; the top Local Set uses AbsWgs84 and the bottom Local Set uses AbsEcef. Producers may not create a Position Local Set including both AbsWgs84 and AbsEcef.

Requirement	
ST 1902-06	Where a class has two or more inheriting child classes only one inheriting child class shall be encoded in the parent class Local Set.

6.1.4 Directed Association (Reference) Transmutation

Directed Associations may be singular (REF<ClassName>) or an array (REF<ClassName>[]).

A single Directed Association or Reference ("REF") is a mimdId value with a two-value Tuple Type. A single Directed Association value transmutes using the same instructions as the Tuple Type in Section 6.1.1.1.7 with the first value equal to the serial identifier and the second value equal to the group identifier. If the group identifier is zero it may be removed from the result.

6.1.4.1 Transmuting an Array of Directed Associations

An array of Directed Associations, or references, transmutes to a list of BER-OID values. Directed Associations or References are mimdIds with a two-value Tuple Type. When transmuting an array of Directed Associations, each mimdId transmutation must contain both the serial number and the group identifier, even if the group identifier is the default group value (zero).

A receiver interprets the list as pairs of values, with the first value of each pair being the serial number, and the second value of the pair being the group id.

The transmutation of an array of Directed Associations transmutes each mimdId, using the instructions in Section 6.1.4, but a group identifier equal to zero are not removed. The resulting values combine into one list of BER-OID values. The Local Set's item length is the number of bytes for the combined list of BER-OID values

An example transmutation of a Directed Association array with three elements is: the first reference has instance serial number = 100 and group identifier = 200; the second reference has serial number = 10 and group identifier= 0; and the third reference has instance serial number = 400 and group identifier = 160000. This becomes a six element UInt list containing the values [100, 200, 10, 0, 400, 160000], which transmutes into the six BER-OID values [0x64, 0x8148, 0x0A, 0x00, 0x8310, 0x89E200], resulting in a single KLV item's value: 0x6481 480A 0083 1089 E200. The KLV item's length for this example is ten bytes.

The decoding process repeatedly decodes individual BER-OID values in pairs from the KLV item's value without exceeding the KLV item's length.

6.2 Class based Transmutation

Future revisions of this standard will support class by class-based transmutation to take advantage of existing MISB Standards. For example, the SDCC class would use MISB ST 1010 to compact the data efficiently. The method of signaling when to use Class based transmutation is under investigation.

6.3 MIMD Packet Instructions

A MIMD Packet contains a KLV key, packet length, payload, and packet error checking. As shown in Figure 13 the KLV key (shown in blue) is the MIMD KLV Defined Length Pack key, which is a 16-byte Universal Label identifying the MIMD Packet as a Defined Length Pack (DLP).

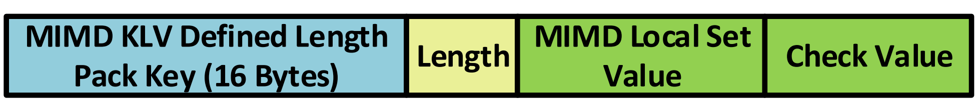


Figure 13: MIMD Packet (DLP)

The packet length (Length shown in yellow) is the length of the payload. The payload has two elements: the MIMD Local Set Value and a Check Value (both shown in green). The MIMD Local Set Value is the KLV encoding of the MIMD class instance and all its child instances resulting from applying the MIMD Model-to-KLV Transmutation Instructions to the MIMD class instance. The MIMD Packet only includes the value portion of the MIMD Class transmutation; that is, there is no Tag nor Length in the MIMD Local Set Value shown in green.

The Check Value is the result of computing a Cyclic Redundancy Check (CRC) on the KLV Key and MIMD Local Set Value. The Check Value is a fixed size, two-byte value.

The MIMD KLV DLP Universal Label is registered in MISB ST 0807 [11] as:

06.0E.2B.34.02.05.01.01.0E.01.05.03.00.00.00.00 (CRC 12649)

As shown in Figure 14, the Check Value is produced by the Check Algorithm on the bytes for the MIMD Defined Length Pack Key and the MIMD class data, which does not include the length[2]. Including the Key ensures no corruption of the key during transmission. The Check Value excludes the length because the length is self-validating with the Check Algorithm; if the length value changes (i.e., due to corruption) the Check Algorithm will have incorrect input and should produce an incorrect Check Value. The MIMD Check Algorithm is the CRC-16-CCITT, which produces a two-byte Check Value. The Motion Imagery Handbook lists the CRC-16-CCITT algorithm in an appendix.

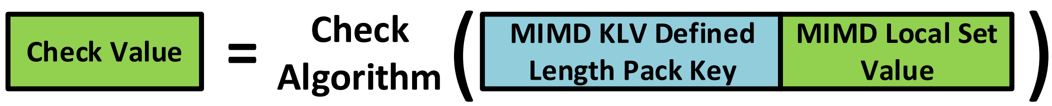


Figure 14: Check Value Computation

Requirement	
ST 1902-07	The CRC-16-CCITT shall be calculated in accordance with the algorithm in the MISB Motion Imagery Handbook.

When parsing a MIMD packet, the length of the MIMD Local Set Value is computed by subtracting two (two bytes for the Check Value) from the DLP's length value.

7 Change Management

As MISB ST 1902 evolves there are two types of possible changes to the MIMD Packet and the MIMD KLV-to-Model Transmutation Instructions: Compatible and Incompatible. Compatible changes are updates to the standard where an existing implementation will continue to function but without necessarily utilizing a newly added capability; these are "backward compatible" changes. An example is adding a new data type to the model which requires an update to the transmutation rules. In this case, MISB ST 1901 requires receivers to ignore unknown [to the receiver] attributes, so the new data type does not affect an existing receiver processing newer metadata.

Incompatible changes are updates to the standard where an existing implementation will not process the data correctly. Examples of incompatible changes include using a different check

[2] The input into the Check Algorithm is different from other MISB standards (e.g., MISB ST 0601). The Check Value is not a Local Set item inside the data to validate, so the CRC value/item does not require special processing.

value algorithm or changing the method of processing arrays. When incompatible changes occur in the standard, the updated standard will define a new MIMD KLV Key for the MIMD Packet.